IML Project Report

Heikki Nenonen, Arttu Koskinen, Niko Petjakko

2022-12-15

Read data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model

npf_test = pd.read_csv("initial_data/npf_test_hidden.csv")
npf_train = pd.read_csv("initial_data/npf_train.csv")
```

Preprocess data & Feature selection

```

npf_train_test = npf_train.set_index("date")
npf_train_test = npf_train_test.drop(['id', 'partlybad'], axis=1)

class2 = np.array(["nonevent", "event"])
class2 = class2[(npf_train_test["class4"]!="nonevent").astype(int)]
#class2 = class2.apply(lambda x: 1 if "event" else 0)
npf_train_test.insert(loc=0, column="class2", value=class2)

npf_train_test["class2"].replace(["event", "nonevent"],[1,0], inplace=True)
npf_train_test["class4"].replace(["nonevent", "Ia", "Ib", "II"],[0, 1, 2, 3], inplace=True)

#DROPS STANDARD DEVIATIONS (STD)
npf_train_test = npf_train_test.filter(regex='mean|class4|class2')
```

As standard deviations didn't affect the accuracy, we decided to not use them for training the models.

First five columns and rows

```
knitr::kable(head(py$npf_train_test[,1:5]), row.names = TRUE, digits = 2)
```

	class2	class4	CO2168.mean	CO2336.mean	CO242.mean
2000-01-17	1	2	368.77	368.67	369.37
2000-02-28	0	0	378.20	378.08	378.67
2000-03-24	1	2	373.04	372.93	373.57
2000-03-30	1	3	375.64	375.55	376.05
2000-04-04	0	0	377.66	377.61	378.12
2000-04-07	1	1	373.87	373.79	374.36

Imports

We chose to compare many of the models used in the course exercise sets

```
from sklearn.dummy import DummyClassifier
from sklearn.dummy import DummyRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import mean_squared_error, accuracy_score

from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier
```

Functions for model loss & accuracy

```
def loss(X_tr, y_tr, X_te, y_te, m):
    return mean_squared_error(y_te, m.fit(X_tr, y_tr).predict(X_te), squared=False)


def accuracy(X_tr, y_tr, X_te, y_te, m):
    return accuracy_score(y_te, m.fit(X_tr, y_tr).predict(X_te))
```

Train, fit, evaluate

```


def magic(models, classtype):

  X = npf_train_test.drop(["class2", "class4"], axis=1, inplace=False)
  y = npf_train_test[classtype]
  X_train, X_test, y_train, y_test = train_test_split(
      X, y, train_size=0.8, random_state=41, shuffle=True, stratify=y
  )

  res = pd.DataFrame(index=models)
  # Loss on training data, for model trained on training data:
  res["train_loss"] = [loss(X_train, y_train, X_train, y_train, m) for m in models]
  # Cross-validation loss:
  res["cv_loss"] = [
      -cross_val_score(
          m, X_train, y_train, cv=10, scoring="neg_root_mean_squared_error"
      ).mean()
      for m in models
  ]
  # Los on test data, for model trained on training data:
```

	train_loss	cv_loss	test_loss	test_accuracy	test_perplexity
Dummy	0.71	0.72	0.71	0.49	2.00
Logistic1	0.37	0.39	0.34	0.88	1.36
Logistic2	0.38	0.39	0.34	0.88	1.37
Logistic3	0.38	0.39	0.34	0.88	1.37
Logistic4	0.38	0.39	0.33	0.89	1.35
Logistic5	0.38	0.39	0.34	0.88	1.37
GausNB	0.41	0.42	0.41	0.83	3.42
QDA	0.19	0.40	0.40	0.84	16.08
RF	**0.00**	**0.33**	**0.29**	**0.91**	**1.26**
KN	0.36	0.43	0.44	0.81	Inf
SVM	0.30	0.36	0.27	0.92	NaN

	train_loss	cv_loss	test_loss	test_accuracy	test_perplexity
Dummy	1.73	1.73	1.74	0.49	1.51
Logistic1	1.08	1.12	1.17	0.63	1.40
Logistic2	1.20	1.23	1.11	0.63	1.52
Logistic3	1.20	1.23	1.12	0.63	1.52
Logistic4	1.08	1.12	1.17	0.63	1.40
Logistic5	1.20	1.23	1.12	0.63	1.53
GausNB	1.31	1.37	1.30	0.54	3.12
QDA	0.54	1.36	1.25	0.63	Inf
RF	**0.00**	**1.03**	**1.04**	**0.71**	**1.43**
KN	0.94	1.17	1.41	0.57	Inf
SVM	0.91	1.02	1.17	0.68	NaN

```
  res["test_loss"] = [loss(X_train, y_train, X_test, y_test, m) for m in models]
  res["test_accuracy"] = [accuracy(X_train, y_train, X_test, y_test, m) for m in models]

  perplexity = lambda p: np.exp(-np.mean(np.log(y_test*p + (1 - y_test) * (1 - p))))

  #temporary solution since svm is weird with perplexity, KEEP SVM LAST!!
  list = [perplexity(m.predict_proba(X_test)[:,1]) for m in models[0:-1]]
  res["test_perplexity"]= np.append(list, np.nan)

  return res
```

Define and compare models

```
results_class2 <- py$results_class2
rownames(results_class2) <- c('Dummy','Logistic1','Logistic2','Logistic3','Logistic4','Logistic5','Gaus
results_class4 <- py$results_class4
rownames(results_class4) <- c('Dummy','Logistic1','Logistic2','Logistic3','Logistic4','Logistic5','Gaus

library(kableExtra)
library(tidyverse)

knitr::kable(results_class2, row.names = TRUE, digits = 2)%>%kable_styling()%>%row_spec(9,bold=T,hline_
```

	train_loss	cv_loss	test_loss	test_accuracy	test_perplexity
gini	**0**	**1.02**	**1.01**	**0.70**	**1.41**
log_loss	0	0.98	1.05	0.73	1.39
entropy	0	1.02	1.04	0.72	1.39
gini n=50	0	1.03	0.98	0.73	1.37
gini n=250	0	1.00	1.00	0.72	1.38
useless	0	1.02	0.96	0.70	NaN

```
knitr::kable(results_class4, row.names = TRUE, digits = 2)%>%kable_styling()%>%row_spec(9,bold=T,hline_a
```

Classification with random forest (James et al.)

Random forest is a combination of bagging (bootstrap + aggregation) multiple decision trees and decorrelating them to explore the model space more thoroughly. Sklearn's RandomForest-Classifier inherently supports multiclass classification, so therefore we could straightforwardly use it for both of the classification problems.

Decision tree is a binary tree that splits the training data, minimizing the impurity at each split as measured by some `criterion`, and ends up in leaf nodes, where only a single class is present. Bootstrap creates `n_estimators` new trees from randomly sampled rows of the training data set. At each split, decorrelation is achieved by sampling m predictors for the splits to use. Finally, the predictions made by the trees are aggregated through majority vote to output the predicted class.

With a sufficient number of trees, the bagging prevents overfitting, but may still get stuck in local optima. The addition of decorrelating avoids this downside, reducing test and OOB (out-of-bag) errors.

Compare different parameters for RandomForestClassifier: impurity measure and number of trees

For the impurity measure (`criterion`) we chose Gini index, and for number of trees (`n_estimators`) 100, as these produced the best accuracy. These just so happen to be the default parameters for the implementation of RandomForest in Sklearn.

```
results_class4 <- py$results_class4
rownames(results_class4) <- c('gini', 'log_loss','entropy','gini n=50', 'gini n=250', 'useless')
knitr::kable(results_class4, row.names = TRUE, digits = 2)%>%kable_styling()%>%row_spec(1,bold=T,hline_a
```

Final model

We first train the model and classify events and nonevents (class2)

```
npf_test_clean = npf_test.drop(['id', 'partlybad', 'date'], axis=1)

#DROPS STDS
npf_test_clean = npf_test_clean.filter(regex='mean|class4|class2')

X = npf_train_test.drop(columns=['class2', 'class4'])
y = npf_train_test['class2']
rfc = RandomForestClassifier(criterion='gini')
model = rfc.fit(X, y)

predict_x = npf_test_clean.drop(columns='class4')
```

```
probas = pd.DataFrame(predict_x.copy())


def get_predict_proba(row, model):
  probas = model.predict_proba(row.values.reshape(1,-1))

  nonevent_p = probas[0][0]
  event_p = 1-nonevent_p

  return event_p

probas['proba'] = probas.apply(lambda row: get_predict_proba(row, model), axis=1)
```

```
predicts = pd.DataFrame(model.predict(predict_x))

final = predicts.merge(probas['proba'].to_frame(), left_index=True, right_index=True)

final[0].replace([0, 1],["nonevent", "event"], inplace=True)
```

Then we train the multiclassifier using only rows that were classified as events

```
npf_test_clean = npf_test.drop(['id', 'partlybad', 'date'], axis=1)

# DROPS STDS
npf_test_clean = npf_test_clean.filter(regex='mean|class4|class2')

# SELECT ROWS WHERE BINARY CLASSIFIER PREDICTED EVENTS AS TRAINING DATA
npf_train_events = npf_train_test[npf_train_test['class2']==1]

X = npf_train_events.drop(columns=['class2', 'class4'])
y = npf_train_events['class4']
rfc = RandomForestClassifier(criterion='gini')
model = rfc.fit(X, y)

predict_x = npf_test_clean.loc[final[final[0]=='event'].index].drop(columns='class4')

predicts = pd.DataFrame(model.predict(predict_x))

predicts[0].replace([1, 2, 3],["Ia", "Ib", "II"], inplace=True)

i = 0
for index, row in final.iterrows():
  if row[0]=='event':
    final.at[index, 0] = predicts.iloc[i,0]
    i += 1
```

Save answers.csv

```
row0 = pd.DataFrame({0: 0.9, 'proba':''}, index =[0])
row1 = pd.DataFrame({0: 'class4', 'proba':'p'}, index =[0])
merged = pd.concat([row1, final])
merged = pd.concat([row0, merged])
```

```
merged.to_csv('answers.csv', index=False, header=False)
```

Conclusions

Why the good accuracy?

Challenge set binary accuracy ≈ 88%, multi ≈ 71%

Random forest has randomness involved and we got lucky, we used the default parameters after all

Why infinite perplexity? Some points in the test set probably got assigned 0% probability (either outliers or then the model is not general enough?)

What could we have done differently?

- More EDA (e.g., seaborn pairwise)
- Would have tried PCA if had time (some tests suggested that it could have improved test accuracy)
- Set random_state for the RF model as well to make the results reproducible
- Use a systematic process to try out different random states for both the data split and the models themselves
- After the model was chosen, we only compared parameters for class4, while we could have also done it for class2

References

Course book by James et al.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

https://scikit-learn.org/stable/modules/multiclass.html

https://www3.nd.edu/~dchiang/teaching/nlp/2016/notes/chapter5v5.pdf (inf perplexity)